МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ УКРАИНЫ
НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
“ХАРЬКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ”

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

к лабораторным работам
«Использование интегрированного языка запросов LINQ»
по дисциплине
«Распределенные информационно-аналитические системы»
для студентов специальностей
122 – Компьютерные науки и информационные технологии,
124 – Системний анализ

Утверждено
редакционно-издательским
советом университета,
протокол № 2 от 23.06.2016 г.

Харьков
НТУ “ХПИ”
2017

Методические указания к лабораторным работам «Использование интегрированного языка запросов LINQ» по дисциплине «Распределенные информационно-аналитические системы» для студентов специальностей 122 – Компьютерные науки и информационные технологии, 124 – Системний анализ / Сост. Ю. Н. Кожин, О. Н. Малых, В. Ф. Прокопенков. – Харьков.: НТУ “ХПИ”, 2017. – 40 с. – На рус. яз.

Составители: Ю.Н. Кожин,
О.Н. Малих,
В.Ф. Прокопенков

Рецензент О.В. Горелый

Кафедра системного анализа и информационно-аналитических технологий

ВВЕДЕНИЕ

Language Integrated Query (LINQ) — проект Microsoft по добавлению языка запросов, напоминающего SQL, в языки программирования платформы .NET Framework.

LINQ обладает расширяемой архитектурой, которая позволяет сторонним разработчикам реализовать доступ к хранилищам данных через механизм запросов.

В отличие от других языков запросов (SQL, QBE и т.п.), которые применяются для разных типов источников данных, LINQ можно использовать для запроса практически любого источника данных. Поэтому разработчикам может понадобиться только один синтаксис запроса.

LINQ предоставляет стандартные шаблоны для работы с данными в различных видах источников и различных форматов. Стандартные шаблоны включают в себя основные операции запросов LINQ: фильтрация, упорядочение, группировка, соединение, выбор (проецирование), статистическая обработка. По форме синтаксис языка LINQ очень похож на язык запросов SQL.

1. ЛАБОРАТОРНАЯ РАБОТА 1
Интегрированный язык запросов LINQ

Цель работы: освоение приемов создания запросов с помощью языка LINQ при работе с массивами данных.

1.1. Запросы LINQ

Запрос **LINQ** (Language-Integrated Query) представляет собой выражение, с помощью которого можно получать данные от источника данных. Запросы обычно выражаются на специальном языке запросов, например, **SQL** для реляционных баз данных и **XQuery** для **XML**.

LINQ – это модель для работы с данными в различных видах источников данных и в различных форматах.

Все операции запроса **LINQ** можно разбить на три различные группы:

1. Получение источника данных.
2. Создание запроса.
3. Выполнение запроса.

1.1.1. Источник данных

Источник данных для выполнения запросов **LINQ** должен поддерживать интерфейс **IEnumerable(Of T)**, где **T** – тип элементов источника данных.

В качестве источника данных могут выступать массивы (списки), базы данных и **XML**-документ.

Для использования массива в качестве источника данных достаточно объявление массива и присваивание значений элементам.

Например, если в качестве источника данных используется массив целых чисел, его объявление может иметь вид:

```
class Array_INT_LINQ
{     static void Main()
   {    int[] numbers = new int[7] { 0, 1, 2, 3, 4, 5, 6 ,7 ,8 ,9};
        ….
   }}
```

Массив numbers может использоваться как источник данных.

Если в качестве источника запросов выступает **XML**-документ, необходимо подключение сборки **System.Xml.Linq**, а также объявление переменной типа **XElement**. Например:

```
using System.Xml.Linq;
XElement contacts = XElement.Load(@"c:\myList.xml");
```

Использование **LINQ** для базы **SQL** требует модуль **System.Data.Linq**. Работа с базой данных построена на основе объектно-реляционного сопоставления классов приложения и таблиц базы данных. Заранее разработанные запросы заполняют списки (массивы) объектов данными из таблиц базы данных.

Для организации связи с базой данных требуется объявление переменной типа DataContext. Например:

```
using System.Data.Linq;
DataContext db = new DataContext (@"c:\northwind\northwnd.mdf");
```

1.1.2. Создание запроса

Запрос указывает, какую информацию нужно извлечь из источника или источников данных. При необходимости запрос также указывает способ сортировки и группировки. Запрос хранится в переменной запроса и инициализируется выражением запроса. В C# используется синтаксис:

тип переменная запроса = **from** *переменная диапазона* **in** *источник*
where *условие отбора данных*
group *группировка*
orderby *сортировка*
select *возвращаемое значение* ;

Части **where, group, orderby** могут отсутствовать.

В качестве типа может быть использован тип **IEnumerable**<…> либо ключевое слово **var**. Предложение **from** указывает источник данных, предложение **where** применяет фильтр, а предложение **select** указывает тип возвращаемых элементов. В **LINQ** переменная запроса хранит сведения, необходимые для предоставления результатов при последующем выполнении запроса.

Например, для массива **numbers** можно задать запрос для выделения всех чётных чисел с помощью объявления

```
IEnumerable<int> evenNumQuery =  from num in numbers
                                 where ((num % 2) = = 0) or (num % 3) = = 0)
                                 select num;
```

либо

```
var evenNumQuery =  from num in numbers
                    where ((num % 2) = = 0) or (num % 3) = = 0)
                    select num;
```

1.1.3. Выполнение запроса

Фактическое выполнение запроса откладывается до выполнения итерации переменной запроса в операторе **foreach**. Эту концепцию называют *отложенным выполнением*. Например, для цикла

```
foreach (int num in numQuery)
{
   Console.Write("{0,1} ", num);
}
```

оператор **foreach** извлекает один элемент результата запроса.

Тип переменной **num** должен соответствовать типу данных источника данных. Например, в предыдущем запросе переменная итерации **num** содержит каждое (по очереди) значение, которое делится на 2 или 3 без остатка.

Запросы, выполняющие статистические функции над диапазоном исходных элементов, должны сначала выполнить выборку этих элементов. Примерами таких запросов являются **Count**, **Max**, **Min**, **Average** , **Last** и **First**. Они выполняются без явного оператора **foreach**. Следующий запрос возвращает количество четных чисел в исходном массиве.

```
var evenNumQuery =
   from num in numbers
   where (num % 2) = = 0
   select num;
int evenNumCount = evenNumQuery.Count();
```

Для вычисления значений функций **Max**, **Min**, **Average**, **Last** и **First** требуется преобразование к простой последовательности с помощью мето-

да **ConvertAll**<*mun*>. Например, для анализа поля года может быть использована следующая последовательность команд:

```
var  StudentQuery6=
      (from st in students select st ).ToList().ConvertAll<Int32>
                  (delegate  (Student r)   { return (r.God); });
        int v1 = (Int32) StudentQuery6.Average();
        int v3 = StudentQuery6.Max();
        int v4 = StudentQuery6.Min();
```

Чтобы принудительно вызвать немедленное выполнение любого запроса и кэшировать его результаты, можно вызвать метод **ToList(Of TSource)** или **ToArray(Of TSource)**. Например:

```
List<int> numQuery2 =
   (from num in numbers  where (num % 2) = = 0  select num).ToList();
```

или

```
var numQuery3 =
   (from num in numbers  where (num % 3) = = 0 select num).ToArray();
```

*1.1.4. Переменная запроса **LINQ***

Переменные запросов **LINQ** определены как **IEnumerable(Of T)** или как производный тип, например **IQueryable(Of T)**. Если переменная запроса имеет тип **IEnumerable< Student >**, это означает, что запрос при выполнении выведет последовательность объектов **Student** (ноль или более элементов). Например:

```
public class Student
{  public string FirstName { get; set; }
   public string LastName { get; set; }
   public int      ID          { get; set; }
  public int      God         { get; set; }
  public List<int> Ball; }
static List<Student> students = new List<Student>
{   new Student {FirstName="Иван", LastName="Дягтерёв", ID=111,
                  Ball= new List<int> {97, 92, 81, 60}},
  new Student {FirstName="Михаил", LastName="Остапенко", ID=122,
                   Ball= new List<int> {94, 92, 91, 91} }
```

```
};
IEnumerable< Student > StudentQuery =
   from st in students  where st. First = = "Иван " select st;
foreach (Student cst in StudentQuery)
{    Console.WriteLine(cst.LastName + ", " + cst.FirstName);  }
```

Вместо объявления типа в запросе можно использовать ключевое слово **var**. Ключевое слово **var** сообщает компилятору о необходимости определения типа переменной запроса с помощью просмотра источника данных, указанного в предложении **from**. Например:

```
var customerQuery2 =
   from st in students  where st. First = = "Иван " select st;
foreach(var cst in StudentQuery)
{   Console.WriteLine(cst.LastName + ", " + cst.FirstName); }
```

Переменная **cst** будет иметь тип поля элемента **st** запроса.

Ключевое слово **var** удобно, когда тип переменной является очевидным или когда не требуется явно указывать тип.

1.1.5. Переменная диапазона запроса

В первую очередь в запросе **LINQ** нужно указать источник данных. В запросе **LINQ** первым идет предложение **from** для указания *источника данных* (**students**) и *переменная диапазона* (**st**).

```
var queryAllStudents =  from st  in  students   select st;
```

Переменная диапазона имеет сходство с переменной итерации в цикле **foreach** за исключением того, что в выражении запроса не происходит фактической итерации. При выполнении запроса переменная диапазона будет использоваться как ссылка на каждый последующий элемент в **students**. Поскольку компилятор может определить тип **st**, нет необходимости указывать его в явном виде.

1.1.6. Фильтрация

Фильтр приводит к возвращению запросом только тех элементов, для которых выражение является истинным. Фильтр задается с помощью части **where**. Фильтр содержит выражение, которое возвращает логическое значение для очередного элемента последовательности. В следующем примере возвращаются записи, в которых поле FirstName равно "Иван".

var customerQuery2 = from st in students where st. FirstName = = "Иван " select st;

Условия отбора элементов можно комбинировать с использованием операции **AND (и)** и **OR (или)**. В C# операции и/или записываются как «&&» и «||» соответственно. Условия можно группировать с помощью скобок. Например:

```
where  (st.FirstName = ="Михаил" || st.LastName = = "Дягтерёв")
&& God=1999
```

Определяет условие отбора студентов 1999 года рождения с именами "Михаил" или с фамилией "Дягтерёв".

1.1.7. Сортировка

Предложение **orderby** обеспечивает выдачу элементов последовательности в зависимости от правила сравнения по умолчанию для сортируемого типа. Например, следующий запрос может быть расширен для сортировки результатов на основе свойства Name. Поскольку Name является строкой, сравнение по умолчанию выполняется в алфавитном порядке.

```
var query3 =  from st in customers  where st.God = = 2001
                    orderby st.LastName ascending   select st;
```

Для упорядочения результатов в обратном порядке используется предложение **orderby**…**descending**.

1.1.8. Группировка

Предложение **group** позволяет группировать результаты на основе указанного ключа. Например, можно указать, что результаты должны быть сгруппированы по **God** так, чтобы все студенты одного года рождения были в одной группе.

```
var queryByGod =   from st in Students  group cust by st.God, FirstName;
foreach (var stGroup in queryByGod)
{  Console.WriteLine(customerGroup.Key);
   foreach (var cst in stGroup)
   {    Console.WriteLine("    {0}", cst.FirstName);  }
}
```

Когда запрос завершается предложением **group**, результаты представляются в виде списка из списков. Каждый элемент в списке является объек-

том, имеющим поле **Key** и список членов группы, сгруппированных по этому ключу. При итерации запроса, создающего последовательность групп, необходимо использовать вложенный цикл **foreach**. Внешний цикл выполняет итерацию каждой группы, а внутренний цикл — итерацию членов каждой группы.

Если необходимо ссылаться на результаты операции группировки, можно использовать ключевое слово **into** для создания идентификатора, который можно будет запрашивать. Следующий запрос возвращает только те группы, которые содержат более двух заказчиков.

```
var custQuery =
   from cust in Students group cust by cust.Last into custGroup
            where custGroup.Count() > 2  orderby custGroup.Key
            select custGroup;
```

1.1.9. Выбор (проецирование) полей объекта

Предложение **select** создает результаты запроса и задает форму или тип каждого возвращаемого элемента. Например, можно указать, будут ли результаты состоять из полных объектов **Students**, только из одного члена, подмножества членов или некоторых других типов данных, на основе вычислений или создания новых объектов. Когда предложение **select** создает что-либо отличное от копии исходного элемента, операция называется *проекцией*.

Для проецирования результата запроса в части **select** реализуют создание нового объекта анонимного класса (класс не имеет собственного имени) по формату:

```
from запрос   select    new { список полей }
```

Например, если из списка **List<Student>** требуется выбрать только поля фамилии и года рождения, то запрос может иметь вид:

```
from st in students   select new { st.LastName, st.God };
```

1.2. Пример использования запроса LINQ

Пусть имеются студенты, которые учатся в различных группах. Каждый студент имеет следующие характеристики: имя, фамилия, год рожде-

ния, номер группы, набор оценок, полученных в сессию. Для хранения информации о студенте может быть использован класс:

```
public class Student
{ public string FirstName { get; set; }
  public string LastName { get; set; }
  public int     ID          { get; set; }
  public string  Ngr         { get; set; }
  public int     God         { get; set; }
  public List<int> Ball;  }
```

Дополнительное поле **ID** добавлено для однозначной идентификации студента.

Для хранения данных обо всех студентах можно использовать список:

```
List<Student> students;
```

Первоначальное заполнение списка студентов может быть осуществлено при создании:

```
students = new List<Student>
{new Student {FirstName="Светлана", LastName="Богатырева", ID=111,
            Ngr = ="ИФ-51в", God=1990, Ball= new List<int> {4, 3, 5, 4}},
  new Student {FirstName="Иван", LastName="Поддубный", ID=122,
            Ngr = ="ИФ-51в", God=1991, Ball= new List<int> {4, 3, 4, 3}},
   new Student {FirstName="Марина", LastName="Шаталина", ID=133,
             Ngr = ="ИФ-51г", God=1990, Ball= new List<int> {5, 5, 5, 5}},
  new Student {FirstName="Петр", LastName="Бержной", ID=144,
              Ngr = ="ИФ-51г", God=1989, Ball= new List<int> {4, 4, 5, 5}}
};
```

Запрос на выдачу данных по заданному значению имени студента (использование фильтра):

```
IEnumerable<Student> StudentQuery =
   from st in students where st.FirstName = = "Светлана" select st;
        foreach (Student cst in StudentQuery)
        {
           Console.WriteLine(cst.LastName + " " + cst.FirstName);
        }
```

Запрос на выдачу всех данных в отсортированном по фамилии виде:

```
            var StudentQuery1 =
      from st in students orderby st.LastName ascending select st;
            foreach (Student cst in StudentQuery1)
            {
               Console.WriteLine(cst.LastName + " " + cst.FirstName + " " +
                                                    cst.God + " " + cst.Ngr);
               foreach (int i in cst.Ball)
                        { Console.Write(i + " "); };
               Console.WriteLine();
            }
```

Использование сложного фильтра по «И»:

```
            IEnumerable<Student> StudentQuery2 =
    from st in students where st.Ngr = = "ИФ- 51г" && st.God>1988 select st;
            foreach (Student cst in StudentQuery2)
            {
               Console.WriteLine(cst.LastName + " " + cst.FirstName);
            }
```

Использование сложного фильтра по «ИЛИ»:

```
    IEnumerable<Student> StudentQuery3 =
from st in students where st.Ngr = = "ИФ-51г" || st.Ngr = = "ИФ-51в"
      select st;
            foreach (Student cst in StudentQuery3)
            {
               Console.WriteLine(cst.LastName + " " + cst.FirstName);
            }
```

Группировка записей по году рождения может быть выполнена с помощью следующего запроса:

```
            var StudentQuery4 =
       from st in students group st by st.God into f select f;
            foreach (var nameGroup in StudentQuery4)
    {
      Console.WriteLine("Key: {0}", nameGroup.Key);
      foreach (var student in nameGroup)
      {
```

```
        Console.WriteLine("\t{0},      {1}",      student.LastName,      student.FirstName);
                } }
```

Следует обратить внимание на тип переменной **StudentQuery4** – **var.** Выбор типа **var** позволяет иметь группы произвольного типа. Наличие **into f** обеспечивает формирование списка по группам. Каждая группа имеет дополнительное поле **Key**.

При необходимости осуществлять выборку только некоторых полей можно применить операцию **new** для формирования нового класса (анонимного) с указанием полей основного класса, которые вошли в новый класс.

```
        var StudentQuery5 =
     from st in students orderby st.LastName ascending select new { st.LastName, st.FirstName, st.God };
    foreach (var cst in StudentQuery5)
        {
       Console.WriteLine(cst.LastName + "\t " + cst.FirstName + "\t " + cst.God);
        }
```

1.3. Задание для выполнения лабораторной работы

Разработать консольное приложение для работы с множеством данных в виде списка (массива).

Разработать запросы к списку с использованием типов **IEnumerable**<…> и **var**:

а) вывод всей информации из списка;

б) с использованием фильтра на отдельное поле и на совокупность полей;

в) в отсортированном виде по одному и нескольким полям;

г) с группировкой данных по одному и нескольким полям;

д) с формированием проекции.

Варианты заданий:

1. Разработать структуру данных для хранения информации о книгах. Книга характеризуется: названием, фамилией автора, стоимостью, датой

издания, издательством, списком инвентарных номеров (книга в нескольких экземплярах).

2. Разработать структуру данных для хранения информации о товарах на складе. Товар характеризуется: названием, стоимостью, количеством (в штуках), списком дат поступления на склад, производителем (наименование фирмы).

3. Разработать структуру данных для хранения информации о вычислительной технике на предприятии. Вычислительная техника характеризуется: названием, стоимостью, вычислительной мощностью, списком лиц, эксплуатирующих вычислительную технику.

4. Разработать структуру данных для хранения информации о проектах, выполняемых на предприятии. Для проекта хранится информация: шифр проекта, наименование проекта, стоимость работ для выполнения проекта, дата начала проекта и дата окончания проекта, список лиц, участвующих в проекте.

5. Разработать структуру данных для хранения информации о кинотеатрах города. Для кинотеатра хранится информация: наименование кинотеатра, вместительность (количество мест), год постройки, ранг кинотеатра (для просмотра видеофильмов, для просмотра широкоформатных фильмов, наличие стереоформатного оборудования и т.п.).

6. Разработать структуру данных для хранения информации о троллейбусных маршрутах города. Для каждого маршрута хранится информация: наименование начальной и конечной остановки, количество троллейбусов на маршруте, время проезда от начала маршрута до конца, список номеров троллейбусов на маршруте.

Контрольные вопросы

1. Для чего используется запрос **LINQ**?
2. Какие основные этапы применения запросов **LINQ**?
3. Что может выступать в качестве источника данных в **LINQ**?
4. Какой интерфейс должен поддерживать источник данных?
5. Какой модуль требуется подключать, если в качестве источника данных используется **XML**-документ, база данных?
6. Какие части содержит запрос **LINQ**? Для чего они используются?

7. Какой оператор обеспечивает фактическое выполнение запроса?
8. Что называется отложенным выполнением запроса?
9. Как принудительно вызвать немедленное выполнение запроса?
10. Для чего используется группировка данных?
11. Какие агрегирующие функции можно использовать для работы с группой?
12. Для чего используется переменная запроса?
13. Как задается фильтр запроса? С помощью каких операций можно создавать сложные фильтры?
14. С помощью какого предложения задается сортировка при выдаче значений от источника данных? Как указывается сортировка по возрастанию и по убыванию?

2. ЛАБОРАТОРНАЯ РАБОТА 2

Создание выходной последовательности элементов в LINQ

Цель работы: получение навыков формирования выходных данных для выполнения запросов с использованием **LINQ**

2.1. Создание выходной последовательности запросов

С помощью запроса **LINQ** можно использовать исходную последовательность в качестве входных данных и изменять ее различными способами для создания новой выходной последовательности:

- изменить порядок элементов в последовательности при помощи сортировки и группировки, не изменяя элементов;

- объединить несколько входных последовательностей в одну выходную последовательность, которая имеет новый тип;

- создать выходные последовательности, элементы которых состоят только из одного или нескольких свойств каждого элемента в исходной последовательности (проецирование);

- создать выходные последовательности, элементы которых состоят из результатов операций, выполняемых над исходными данными;

- создать выходные последовательности в другом формате. Например, можно преобразовать данные из строк **SQL** или текстовых файлов в **XML**.

Выходные последовательности одного запроса могут использоваться как входные последовательности для нового запроса.

2.1.1. Сортировка и группировка

При необходимости получать элементы выходной последовательности в определенном порядке может быть выполнена сортировка.

Сортировка данных осуществляется в зависимости от правила сравнения по умолчанию для сортируемого типа.

Для сортировки последовательности по одному или нескольким ключам используется оператор **orderby**:

orderby *поле1 направление сортировки, поле2 направление сортировки …*

После ключевого слова **orderby** через запятую может быть указано одно или несколько полей элемента последовательности с указанием направления сортировки: **ascending** – по возрастанию (по умолчанию), **descending –** по убыванию.

Например, следующий запрос может быть расширен для сортировки результатов на основе свойства **Name** (сравнение выполняется в лексикографическом порядке).

```
var queryLondonCustomers3 =   from cust in customers
                    where cust.City = = "London"  orderby cust.Name ascending
                      select cust;
```

Группировка данных позволяет изменить порядок в выходной последовательности так, что в одну группу будут входить элементы с одинаковым значением поля группировки.

group *переменная диапазона* **by** *поле группировки* **into** *имя группы*.

Имя группы может использоваться в части **where select** запроса вместо переменной диапазона.

```
var queryLastNames =
      from student in students
        group student by student.LastName into newGroup
        orderby newGroup.Key  select newGroup;
```

При выполнении запроса элементы каждой группы выдаются в одном экземпляре. Для каждой группы может также создоваться новый элемент на основе групповой операции.

Когда запрос завершается предложением **group**, результаты представляются в виде списка из списков. Каждый элемент в списке является объектом, имеющим член **Key** и список членов группы, сгруппированных по этому ключу. При итерации запроса, создающего последовательность групп, необходимо использовать вложенный цикл **foreach**. Внешний цикл выполняет итерацию каждой группы, а внутренний цикл — итерацию членов каждой группы.

```
var queryCustomersByCity =
   from cust in customers
   group cust by cust.City;
```

```
foreach (var customerGroup in queryCustomersByCity)
{   Console.WriteLine(customerGroup.Key);
    foreach (Customer customer in customerGroup)
    {    Console.WriteLine("    {0}", customer.Name);   }
}
```

2.1.2. Слияние нескольких входных последовательностей в одну выходную

Запрос **LINQ** можно использовать для создания выходной последовательности, содержащей элементы из нескольких входных последовательностей. В следующем примере показано объединение двух находящихся в памяти структур данных, но те же принципы могут применяться для соединения данных из источников **XML**, **SQL** или **DataSet**. Предположим, что существуют два следующих типа классов:

```
class Student
{   public string FirstName { get; set; }
    public string LastName {get; set;}
    public int ID { get; set; } }
class Teacher
{   public string First { get; set; }
    public string Last { get; set; }
    public int ID { get; set; }
    public string Book { get; set; }}
static List<Student> students = new List<Student>
{ new Student {FirstName="Светлана", LastName="Богатырева", ID=1},
  new Student {FirstName="Иван", LastName="Поддубный", ID=1},
  new Student {FirstName="Марина", LastName="Шаталина", ID=2},
  new Student {FirstName="Петр", LastName="Бережной", ID=3}
};
static List< Teacher > teachers = new List< Teacher >
{ new Teacher{  First= "Олег" ,Last="Козлов"   ,ID=1, Book="Физика"  },
 new Teacher{  First= "Павел",Last="Петров" ,ID=1, Book="Химия"     },
 new Teacher{ First= "Юра",Last="Иванов",ID=1, Book="Математика"},
 new Teacher{ First= "Ира" ,Last="Сидорова" ,ID=1, Book="История" }
```

};

Тогда с помощью операции **Concat** можно объединить результаты двух запросов в один. При объединении следует учитывать, что оба подзапроса должны формировать одинаковый тип данных.

```
var StudentQuery7_1 =  from st in students orderby st.LastName ascending
                  where st.God= =1990  select st;
    var StudentQuery7_2 =  from st in students orderby st.LastName ascending
                  where st.God= =1991 select st;
 foreach (var a in StudentQuery7_1.Concat(StudentQuery7_2 ))
        {  Console.WriteLine("\t{0}, {1}", a.LastName, a.FirstName);  }
```

Если источники данных имеют различную структуру, можно использовать анонимный тип для формирования выходной последовательности. Например:

```
var StudentQuery7 = from st in students orderby st.LastName ascending
                  select new { st.FirstName , st.LastName};
var StudentQuery8 = from th in teachers orderby th.Last ascending
        select new { FirstName=th.First, LastName=th.Last };
 foreach (var a in StudentQuery7.Concat(StudentQuery8))
    {   Console.WriteLine("\t{0}, {1}", a.LastName, a.FirstName);  }
```

2.1.3. Выбор подмножества каждого исходного элемента

Существует два основных способа выбора подмножества каждого элемента в исходной последовательности: выбор одного поля или нескольких полей.

Чтобы выбрать только один член исходного элемента, используется операция «.» (точка).

Например, если объект **Teachers** содержит несколько свойств, то выбрать в качестве результата только свойство **Book** можно, указав в части **select** требуемое свойство.

```
var query = from cust in teachers select cust.Book;
```

Для создания элементов, содержащих более одного свойства исходного элемента, требуется использовать инициализатор объектов с именованным объектом либо с анонимным типом. Например:

```
var query = from cust in teachers
        select new { Teach = cust.Name, Lesson = cust.Book};
```

2.1.4. Выполнение операций над исходными элементами

Выходная последовательность может быть результатом вычислений с использованием исходных элементов в качестве входных аргументов. Например:

```
 double[] rad = { 1, 2, 3 };
 var query_grad = from r in rad  select String.Format("Grad = {0}", (r * 180/ * 3.14));
        foreach (string s in query)
           Console.WriteLine(s);
```

2.1.5. Создание вложенных запросов

В следующем примере показано, как создавать вложенные группы в выражении запроса **LINQ**. Каждая группа, созданная в соответствии с годом, затем подразделяется на группы на основе имен учащихся:

```
 var queryNestedGroups =
 from student in students group student by student.God into newGroup1
    from newGroup2 in  (from student in newGroup1
                                group student by student.LastName)
    group newGroup2 by newGroup1.Key;
foreach (var outerGroup in queryNestedGroups)
    {  Console.WriteLine("DataClass.Student God = {0}",
                                                outerGroup.Key);
       foreach (var innerGroup in outerGroup)
       { Console.WriteLine("\tNames that begin with: {0}",
                                                innerGroup.Key);
          foreach (var innerGroupElement in innerGroup)
          { Console.WriteLine("\t\t{0} {1}", innerGroupElement.LastName,
                    innerGroupElement.FirstName);
          } } }
```

2.1.6. Соединение последовательностей

Простое внутреннее соединение, которое сопоставляет элементы из двух источников данных на основании простого ключа, осуществляется с помощью оператора **join.** Например, для слияния последовательности students и teachers по полю ID, можно использовать запрос:

```
var query = from person   in students  join
                    th_person  in teachers on person.ID equals th_person.ID
          select new{ PName= person.FirstName, TName = th_person.First };
```

2.2. Задание для выполнения лабораторной работы

Разработать консольное приложение для создания последовательностей на основе других последовательностей.

Варианты заданий:

1. Разработать последовательности для хранения данных:

кинофильм – название, год выпуска, жанр, режиссёр;

спектакль – название, автор, жанр, учётный номер актёра главной роли;

актеры – фамилия, имя, отчество, год рождения, амплуа, учетный номер.

Построить запросы:

а) выдать данные из последовательности «кинофильм» в отсортированном по годам виде;

б) сгруппировать данные в последовательности «актёры» по амплуа и по году рождения;

в) слить последовательности «кинофильм» и «спектакль» и выбрать в новую последовательность: название, жанр, режиссер/автор;

г) соединить последовательности «спектакль» и «актёры» и выдать название спектакля, фамилию автора, фамилию актёра в главной роли.

2. Разработать последовательности для хранения данных:

товар – название, стоимость, количеством (в штуках), производитель (шифр);

склад – название, общая площадь, количество автопогрузчиков, год постройки;

производитель – название фирмы, руководитель, шифр.

Построить запросы:

а) выдать данные из последовательности «товар» в отсортированном по стоимости виде;

б) сгруппировать данные в последовательности «склад» по количеству автопогрузчиков и по году постройки;

в) сформировать запрос для подсчета общего числа погрузчиков на складах;

г) соединить последовательности «товар» и «производитель» по полю «шифр» и выдать данные: наименование товара, фирма производитель.

3. Разработать последовательности для хранения данных:

река – название, длина, максимальная ширина, число притоков, шифр реки;

море – название, площадь, шифр моря;

корабль – название, число пассажиров, пароходство, год постройки, тип (речной, морской), шифр реки/моря.

Построить запросы:

а) выдать данные из последовательности «река» в отсортированном по длине виде;

б) сгруппировать данные в последовательности «корабль» по пароходству и по году постройки;

в) сформировать запрос для подсчета числа кораблей в пароходстве;

г) соединить последовательности «корабль» и «река» по полю «шифр» и выдать данные: наименование корабля, название реки.

4. Разработать последовательности для хранения данных:

самолет – тип самолета, грузоподъемность, максимальная дальность, размах крыльев, длина разбега, шифр компании;

вертолет – тип вертолета, грузоподъемность, максимальная высота, дальность полета, шифр компании;

авиакомпания – название, место размещения офиса, дата образования фирмы, шифр компании.

Построить запросы:

а) выдать данные из последовательности «самолет» в отсортированном по грузоподъёмности виде;

б) сгруппировать данные в последовательности «вертолет» по максимальной высоте подъема и дальности полета;

в) сформировать запрос для подсчета числа самолетов в авиакомпании;

г) соединить последовательности «самолет» и «авиакомпания» по полю «шифр» и выдать данные: наименование самолета, название авиакомпании.

5. Разработать последовательности для хранения данных:

жилой дом – тип проекта, число этажей, число подъездов, дата постройки, шифр;

район города – название, адрес районной администрации, количество жителей, площадь, шифр;

список домов – шифр района, шифр дома.

Построить запросы:

а) выдать данные из последовательности «жилой дом» в отсортированном по количеству этажей виде;

б) сгруппировать данные в последовательности «жилой дом» по типу проекта подъема и количеству этажей;

в) сформировать запрос для подсчета числа домов в районе;

г) соединить последовательности «жилой дом», «район города» и «список домов» по полю «шифр» и выдать данные: наименование района, тип дома, дата постройки.

6. Разработать последовательности для хранения данных:

грузовой автомобиль – марка автомобиля, грузоподъемность, дата выпуска, дата капитального ремонта, государственный номер, шифр автопарка;

такси – марка автомобиля, количество посадочных мест, дата выпуска, государственный номер, шифр автопарка;

автопарк – название, адрес размещения, площадь для размещения автомобилей, шифр.

Построить запросы:

а) выдать данные из последовательности «грузовой автомобиль» в отсортированном по дате выпуска виде;

б) сгруппировать данные в последовательности «такси» по марке автомобиля и по дате выпуска;

в) сформировать запрос для подсчета количества такси в автопарке;

г) соединить последовательности «грузовой автомобиль» и «автопарк» по полю «шифр» и выдать данные: наименование автопарка, тип автомобиля, государственный номер автомобиля.

Контрольные вопросы

1. Какие изменения можно выполнять над исходными последовательностями для получения новых последовательностей?
2. Изменяет ли группировка исходную последовательность или влияет только на выходную последовательность?
3. Как задается имя группы?
4. Где может быть использовано имя группы?
5. Как осуществить перебор групп и элементов в группе для выходной последовательности?
6. Какой метод позволяет добавить в одну последовательность другую?
7. Каким требованиям должны удовлетворять последовательности для их слияния в одну последовательность?
8. С помощью кокой операции осуществляется выборка элементов объекта последовательности для формирования подмножества полей?
9. Что понимают под вложенными запросами?
10. Как формируется запрос на соединения двух последовательностей?

3. ЛАБОРАТОРНАЯ РАБОТА 3
Использование LINQ SQL для работы с базой данных

Цель работы: получение навыков применения запросов с использованием **LINQ** для реализации заранее подготовленных запросов к реляционной базе данных.

3.1. Запросы LINQ SQL для работы с базой данных

Работа с базой данных с использованием **LINQ** построена на основе объектно-реляционного сопоставления классов приложения и таблиц базы данных.

LINQ для **SQL** используется для связи с базой данных и служит источником данных для работы экранной формы. Заранее разработанные запросы заполняют списки (массивы) объектов данными из таблиц базы данных (рис. 3.1). Обрабатывающие процедуры приложения осуществляют выборку данных из списков объектов, не затрагивая базу данных.

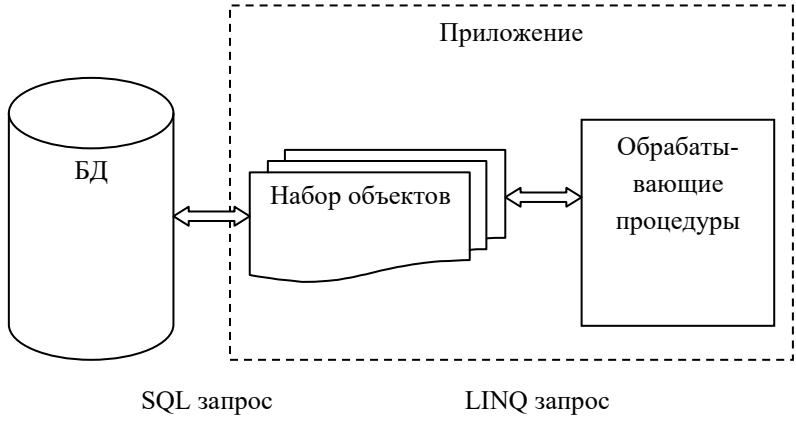

Рисунок 3.1 – Двухзвенная работа с базой данных

3.2. Создание набора классов для работы с таблицами базы данных

Классы для создания набора объектов работы с базой данных могут быть разработаны в режиме конструктора или путем непосредственного программирования.

Реляционный конструктор объектов содержит область визуального конструктора для создания и изменения классов **LINQ** для **SQL** (классов сущностей).

Приложение для работы с базой данных на основе **LINQ** для **SQL** строится следующим образом:

1) добавление файла **.dbml** в проект для связывания с базой данных;
2) создание классов, которые сопоставляются таблицам в базе данных;
3) создание источника данных, который ссылается на классы;
4) создание формы, содержащей элементы управления, которые привязаны к классам;
5) добавление кода для загрузки и сохранения данных от классов сущностей и базой данных;
6) создание запросов **LINQ** и отображение данных на форме;
7) добавление хранимых процедур для вставки, обновления и удаления данных;
8) конфигурирование классов сущностей для использования хранимых процедур для выполнения операций вставки, обновления и удаления данных.

*3.2.1. Добавление файла dbml **LINQ** для **SQL** классов*

Классы сущностей создаются и хранятся в файлах классов **LINQ SQL** (**DBML**-файлы).

Для добавления файла **.dbml** в проект необходимо:

1. В меню «*Проект»* выбрать команду «*Добавить новый элемент*».
2. Выбрать шаблон **LINQ to SQL Classes** (рис.3.2) и ввести имя **.dbml** файла в поле «*Имя»*.

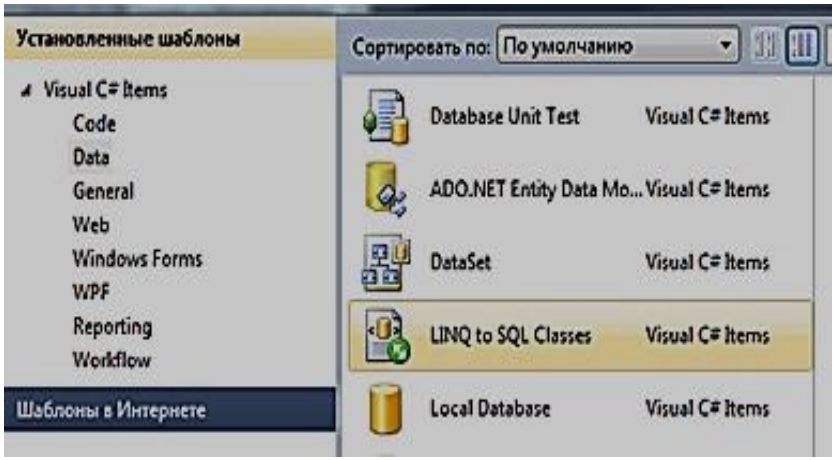

Рисунок 3.2 – Добавление элемента SQL to SQL Classes

После добавления нового файла открывается пустая область конструктора, отображающая две отдельных части. Левая часть – это область сущностей, в которой отображаются и конфигурируются классы сущностей. Часть справа – это область методов **DataContext** (хранимые процедуры базы данных).

С помощью «*Обозревателя серверов*» можно выбрать и подключиться к требуемой базе данных для выбора таблиц и хранимых процедур.

3.2.2. Создание классов сущностей

Создание **LINQ SQL** классов, которые соответствуют таблицам базы данных, осуществляется путем перетаскивания таблиц из *«Обозревателя сервера/обозревателя базы данных»* в область конструктора (рис 3.3) . Набор классов может содержать несколько классов для различных таблиц базы данных.

Каждый класс имеет набор свойств. Основными свойствами для класса являются режим доступа **Access**, источник данных – **Source** и имя класса – **Name** (табл. 3.1).

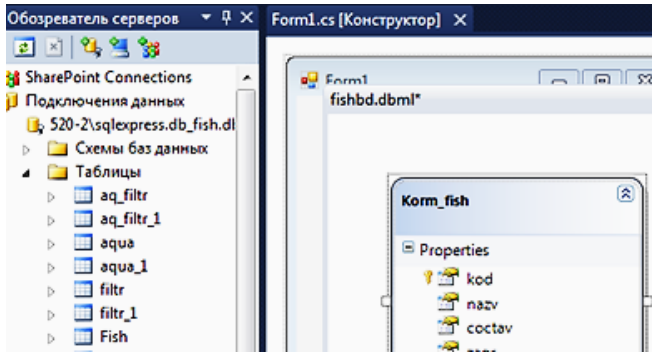

Рисунок 3.3 – Добавление таблиц базы данных и формирование класса

Таблица 3.1 – Свойства классов **LINQ SQL**

Свойство	Значение	Описание
Access	Public, Internal	Тип доступа
Inheritans	None, Abstract, Sealed	Режим наследования
Name		Имя класса
Source	Имя таблицы в БД	Источник (имя таблицы)
Insert		Процедура ввода
Delete		Процедура удаления
Update		Процедура изменения

Каждый класс имеет набор полей, соответствующий колонкам таблицы базы данных. Значения свойств полей базы данных система программирования устанавливает в соответствии с объявлениями в базе данных.

В табл. 3.2 представлены основные свойства атрибутов (полей) класса.

Таблица 3.2 – Основные свойства полей класса

Свойство	Описание
Access	Область видимости (доступ)
Delay Loaded	Отложенная загрузка значения
Name	Имя поля
Type	Тип значений в рамках C#

Продолжение таблицы 3.2

Свойство	Описание
Auto Generated Value	Указывает среде CLR на необходимость сгенерировать значение автоматически (автоинкрементное значение)
Auto-Sync	Указывает среде CLR на необходимость получить значение после операции вставки или обновления. Значения: Always, Never, OnUpdate, OnInsert
Nulable	Указывает, что столбец может содержать значения NULL
Primary Rty	Значение поля является первичным ключом
Read Only	Поле содержит неизменяемое значение
Server Data Type	Тип данных соответствующего столбца таблицы базы данных
Source	Имя столбца таблицы базы данных, соответствующее полю класса
Update Check	Проверка при изменении значения

*3.2.3. Создание источника данных на основе классов **LINQ SQL***

Классы сущностей можно использовать в качестве источников данных. Их можно добавить в окно «*Источники данных*» и перенести на формы для создания связанных с данными элементов управления.

Для добавления в качестве источника данных необходимо выполнить следующие действия:

1) с помощью меню «*Build»* построить проект;

2) в меню «*Данные»* выбрать команду «*Показать источники данных»*;

3) в окне «*Источники данных»* выбрать «*Добавить новый источник данных»*;

4) на странице «*Выбор типа источника данных»* выбрать «*Объект»*;

5) раскрыть *узел*, соответствующий построенному ранее классу **LINQ SQL** (рис. 3.4). Имя класса совпадает с именем файла DBML. Если необходимый класс отсутствует, его требуется пересоздать заново и перестроить проект;

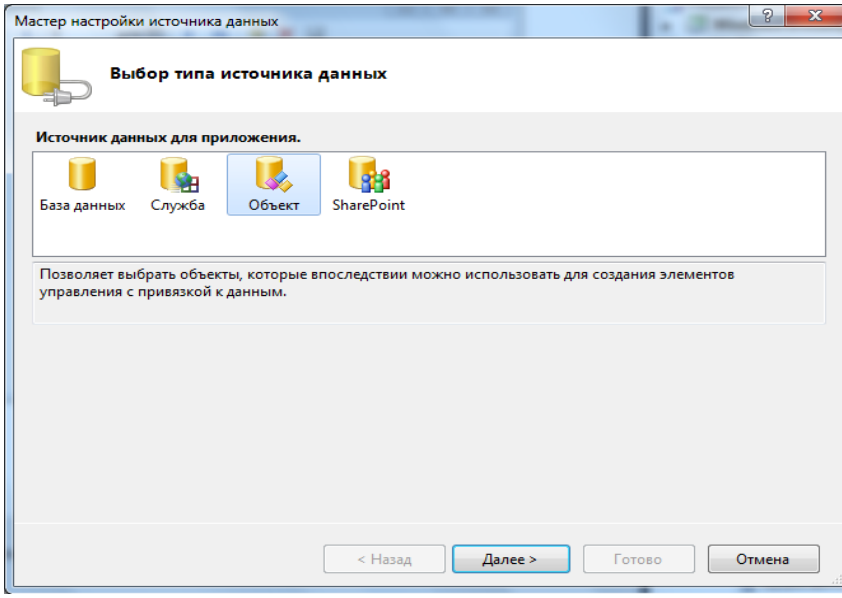

Рисунок 3.4– Выбор источника данных

6) выбрать *необходимые* элементы для работы с источниками данных (рис.3.5);

7) из созданной вкладки перенести на форму необходимые элементы для отображения данных.

3.2.4. Создание элементов управления, связанных с данными

Для визуализации данных можно создать элементы отображения путем перетаскивания соответствующих источников на экранную форму. Для добавления элементов управления, привязанных к классам **LINQ SQL**, необходимо:

1) открыть форму в режиме конструктора;

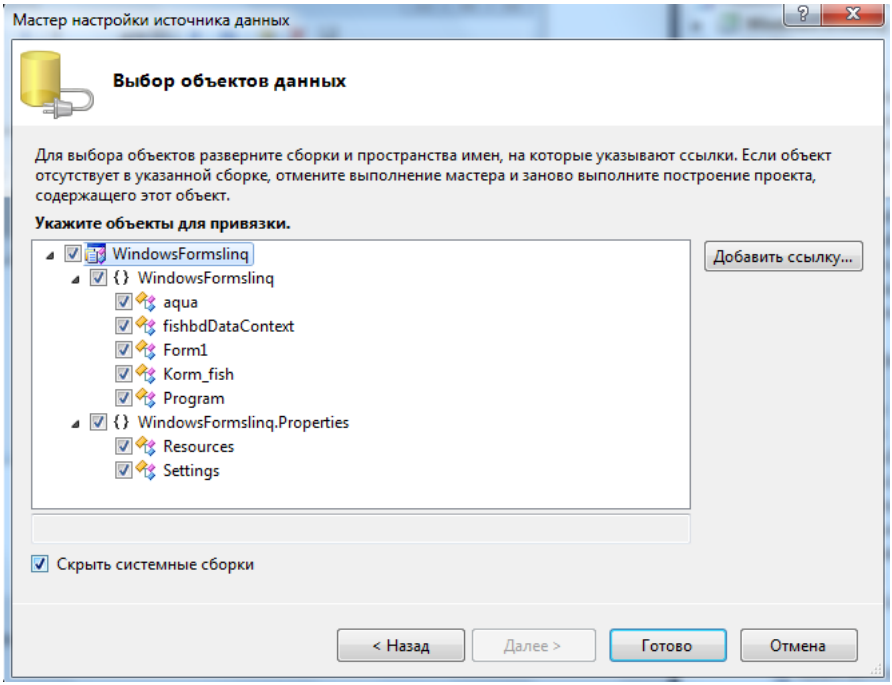

Рисунок 3.5 – Выбор объекта в качестве источника данных

2) из окна «*источники данных*» перетащить узел, соответствующий элементу для отображения на форму;

3) открыть форму в представлении кода;

4) добавить в объявлении формы переменную для организации связи с помощью **linq** для **sql**.

Например:

private FishDataContext fdb = new FishDataContext();

Следует иметь в виду, что тип **FishDataContext** создается автоматически по имени файла **DBML**. Если в приложении несколько наборов классов **LINQ** для **SQL**, необходимо выбрать требуемый тип.

Загрузка данных может быть реализована в обработчике события для **Form_Load**. Для этой цели добавить в обработчик следующий код:

aquaBindingSource.DataSource = fdb.Customers;

После связывания источника данных при работе приложения будут отображаться данные в соответствии с источником данных (рис.3.6).

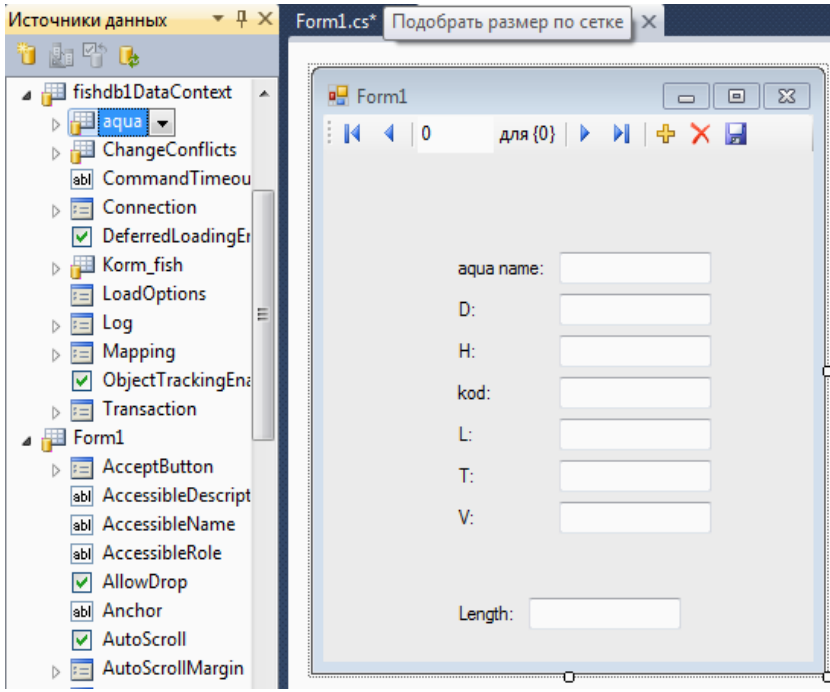

Рисунок 3.6 – Окно приложения

По умолчанию кнопка «*Сохранить*» недоступна, и сохранение изменений не осуществляется. При необходимости добавления режима сохранения требуется для управляющего элемента задать обработчик события с кодом:

```
private void aquaBindingNavigatorSaveItem_Click(object sender, EventArgs e)
 { try
   {
 fdb.SubmitChanges(); // сохранение изменений
```

```
    }
  catch (Exception ex)
  { MessageBox.Show( ex.Message); }
  }
```

*3.2.5. Привязка к запросам **LINQ***

Элементы отображения данных можно привязывать напрямую к запросам **LINQ**. Для этой цели на экранной форме можно создать элементы для ввода условий отбора записей из таблиц базы данных и элемент исполнения запроса (например, кнопка или пункт меню).

Для привязки запроса **LINQ** в обработчик события элемента исполнения запроса можно добавить код объявления запроса **LINQ** и указать источник данных для отображения информации.

```
private void button1_Click(object sender, EventArgs e)
 { var aquaQuery = from aqua1 in fdb.aqua
                where aqua1.aqua_name = = this.aquaBox1.Text
                select aqua1;
 aquaBindingSource.DataSource = aquaQuery;
}
```

3.2.6. Переопределение операций Insert, Update и Delete

При внесении изменений в набор данных приложение фиксирует эти изменения в локальной копии. По окончании работы с набором данных может потребоваться сохранить изменения в базе данных. По умолчанию логика для выполнения обновлений предоставляется средой выполнения **LINQ** для **SQL** через стандартные инструкции **Insert, Update** и **Delete** на основании инструкции **Select**, которая используется для заполнения класса **LINQ** данными. Однако, если выборка данных осуществляется с использованием проекции (не все поля таблицы участвуют в выборке данных) или если класс **LINQ** соответствует представлению (**View**), может потребоваться использование хранимой процедуры базы данных. В этом случае пользователь должен разработать три хранимые процедуры для выполнения операций вставки, удаления и внесения изменений в таблицы базы данных.

При работе с локальным набором данных приложение автоматически устанавливает признаки изменения для каждой записи – добавлена, измене-

на или удалена текущая запись. При выполнении операции сохранения изменений приложение автоматически будет вызывать соответствующую хранимую процедуру с передачей значений полей набора данных через параметры процедуры.

Для переопределения поведения по умолчанию при обновлении необходимо:

1) разработать набор хранимых процедур для таблиц и представлений базы данных, которые будут использоваться в качестве самостоятельных источников данных;

2) открыть файл **linq** для **sql** через «*обозреватель решений*»;

3) в узле «*сохраненные процедуры*» соответствующей базы данных выбрать необходимые процедуры и перенести их в область конструктора (рис.3.7);

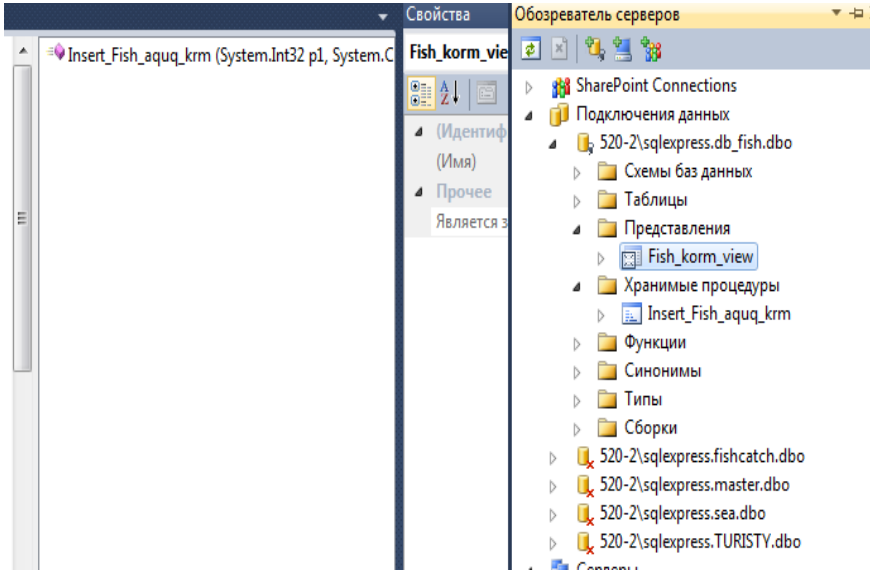

Рисунок 3.7 – Окно приложения

4) выбрать класс сущностей **LINQ** и для соответствующей операции в свойствах класса выбрать одну из хранимых процедур;

5) проверить список аргументов и сопоставить с аргументами хранимой процедуры поля набора данных **LINQ** (рис. 3.8).

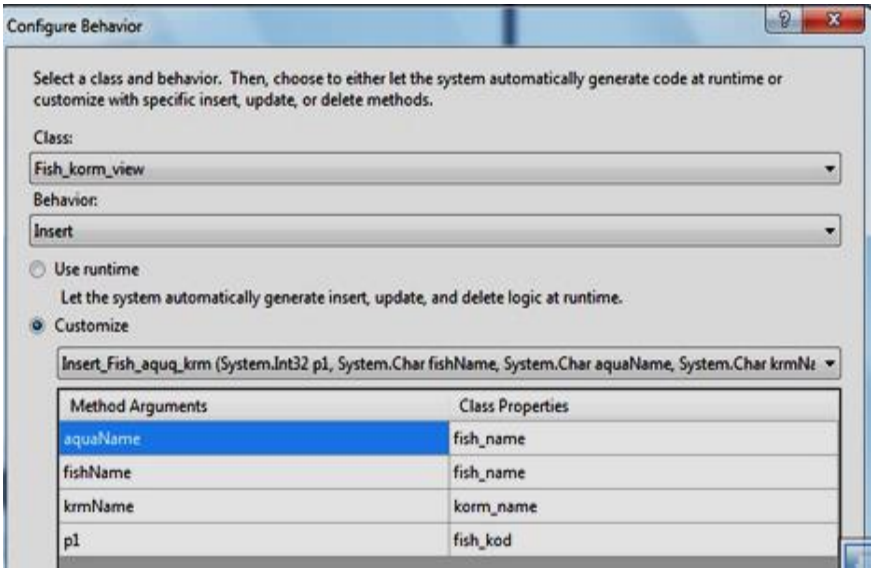

Рисунок 3.8 – Сопоставление с аргументами хранимой процедуры полей набора данных

Сохраненная процедура добавляется в область методов как метод **DataContext**.

3.3. Задание для выполнения лабораторной работы

Разработать оконное приложение для работы с базой данных.

1) Разработать схему базы данных в соответствии с заданием. Добавить в схему представление для вывода полной информации.

2) Для выполнения операций ввода, удаления и внесения изменений в базу данных через представление разработать хранимые процесдуры.

3) С использованием LINQ SQL разработать набор классов для работы с базой данных на основе двухзвенной технологии.

4) Разработать набор стандартных запросов к набору классов с использованием LINQ.

5) Протестировать работу приложения.

6) Оформить отчет

Варианты заданий:

1. Разработать схему базы данных для хранения информации:

кинофильм – название, год выпуска, жанр, режиссёр;

спектакль – название, автор, жанр;

актеры – фамилия, имя, отчество, год рождения, амплуа.

Добавить в таблицы необходимые поля для связи с учетом того, что каждый актер может быть задействован в нескольких спектаклях и кинофильмах.

Разработать обзор для отображения данных: актер, название спектакля или кинофильма.

2. Разработать схему базы данных для хранения данных:

товар – название, стоимость, количеством (в штуках);

склад – название, общая площадь, количество автопогрузчиков, год постройки;

производитель – название фирмы, руководитель.

Добавить в таблицы необходимые поля для связи с учетом того, что каждый товар может производиться различными фирмами, каждая фирма может производить различные товары. На складе могут храниться несколько различных товаров, но отдельный товар может храниться только на одном складе.

Разработать представление для отображения данных: товар (наименование, количество), фирма-производитель (наименование), склад (наименование).

3. Разработать схему базы данных для хранения данных:

корабль – название, число пассажиров, пароходство, год постройки, тип (речной, морской);

река – название, длина, максимальная ширина, число притоков;

море – название, площадь.

Добавить в таблицы необходимые поля для связи с учетом того, что каждый корабль может эксплуатироваться только на реке или только на море. На реке и на море могут эксплуатироваться несколько кораблей.

Разработать представление для отображения данных: корабль (наименование, количество пассажиров, год постройки), название реки/моря.

4. Разработать схему базы данных для хранения данных:

самолет – тип самолета, грузоподъемность, максимальная дальность, размах крыльев, длина разбега;

вертолет – тип вертолета, грузоподъемность, максимальная высота, дальность полета;

авиакомпания – название, место размещения офиса, дата образования фирмы.

Добавить в таблицы необходимые поля для связи с учетом того, что каждая компания может иметь самолеты и вертолеты, но отдельный самолет или вертолёт может принадлежать только одной компании. Имеется несколько самолетов и вертолетов одного типа. Отдельный самолет и вертолет имеет бортовой номер (уникальное значение).

Разработать представление для отображения данных: транспортное средство (бортовой номер, тип), наименование компании, грузоподъемность, максимальная высота, дальность полета.

5. Разработать схему базы данных для хранения данных:

жилой дом – тип проекта, число этажей, число подъездов;

район города – название;

улица домов – наименование.

Добавить в таблицы необходимые поля для связи с учетом того, что на улице может находиться несколько домов. Отдельный дом имеет дату постройки, находится только на одной улице и принадлежит только одному району. Дома на отдельной улице пронумерованы.

Разработать представление для отображения данных: район, улица, номер дома, тип проекта, дата постройки.

6. Разработать схему базы данных для хранения данных:

грузовой автомобиль – марка автомобиля, грузоподъемность;

такси – марка автомобиля, число посадочных мест;

автопарк – название, адрес размещения.

В автопарке имеются или только грузовые автомобили или только такси. Отдельный грузовой автомобиль и такси имеют государственный номер и дату выпуска.

Разработать представление для отображения данных: транспорт (тип, государственный номер, дата выпуска), автопарк.

Контрольные вопросы

1. Что понимают под объектно-реляционным соответствием классов приложения и таблиц базы данных?
2. Что такое двухзвенная технология работы с базой данных?
3. Из каких этапов состоит процесс разработки приложения работы с базой данных на основе LINQ SQL ?
4. Для чего служит файл .dbml?
5. Какие основные свойства классов LINQ SQL?
6. Как создается источник данных на основе LINQ SQL?
7. С помощью какого приема обеспечивается визуализация данных источника на экранной форме?
8. Какие два основных подхода к отображению данных из таблиц базы данных?
9. С помощью какого метода обеспечивается сохранение изменений в базе данных?
10. Как можно осуществить привязку элементов отображения данных с запросом к базе данных?
11. В каком случае может потребоваться переопределение операций вставки, удаления и внесения изменений в таблицы базы данных?
12. Как обеспечивается соответствие полей набора данных и параметров хранимых процедур при выполнении операций вставки, удаления и снесения изменений?

Список литературы

1. Фримен А. LINQ: язык интегрированных запросов в C# 2010 для профессионалов /Адам Фримен, Джозеф С. Раттц-мл. – М.: Вильямс, 2011. 656 с.

2. Албахари Д. LINQ. Карманный справочник /Джозеф Албахари, Бен Албахари. – СПб.: БХВ-Петербург, 2009

3. Зиборов В. Visual C# 2010 на примерах / Виктор Зиборов. – СПб.: БХВ-Петербург, 2011

4. Шилдт Г. C# 3.0. Полное руководство / Герберт Шилдт. – М.: Вильямс, 2010.

Содержание

Навчальне видання

Методичні вказівки
до лабораторних робіт
«Використання інтегрованої мови запитів LINQ»
з дисципліни
«Розподілені інформаційно-аналітичні системи»

Російською мовою

Укладачі:
КОЖИН Юрій Миколайович
МАЛИХ Олег Миколайович
ПРОКОПЕНКОВ Володимир Пилипович

Відповідальний за випуск *О.С.Куценко*

Роботу до друку рекомендував *О.В.Горілий*
Редактор *О.С. Самініна*

План 2016, поз. 86

Підп. до друку 23.03.2017	Формат 60x84 1/16.	Папір офсетний.
Riso–друк.	Гарнітура Таймс.	Ум. друк. арк. 2,3.
Наклад 25 прим.	Зам. №	Ціна договірна.

Видавничий центр НТУ “ХПІ”
вул.Кирпичова, 21, м. Харків, 61002
Свідоцтво суб’єкта видавничої справи ДК №3657 від 24.12.2009 р

ООО Планета Прінт